22. Searching a List

Topics:

- Linear Search
- Binary Search
- Measuring Execution Time
- The Divide and Conquer Framework

Search

Examples:

Is this song in that playlist?

Is this number in that phone book?

Is this name in that phone book?

Is this fingerprint in that archive of fingerprints?

Is this photo in that yearbook?

More on Using Phone Books

The Manhatten phone book has 1,000,000+ entries.

How is it possible to locate a name by examining just a tiny , tiny fraction of those entries?

There must be a great search algorithm behind the scenes.

Linear Search

LinSearch: The Spec

```
def LinSearch(x,a):
    """ Returns an int k with the
    property that a[k]==x is True.
    If no such k exists, then
    k==-1.

    PreC: a is a nonempty list of
    ints and x is an int.
    """
```

Could also apply the same ideas for searching a list of strings.

Linear Search

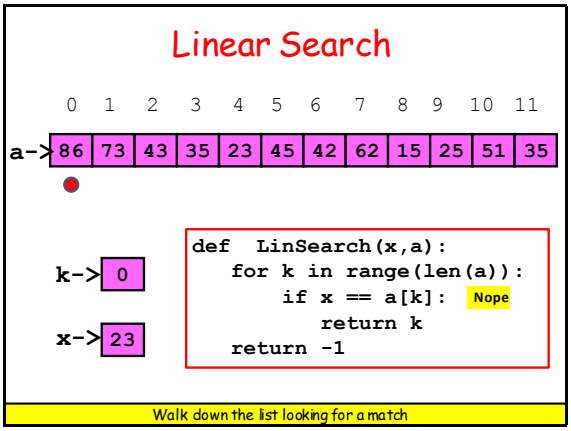
	0	1	2	3	4	5	6	7	8	9	10	11
a->	86	73	43	35	23	45	42	62	15	25	51	35

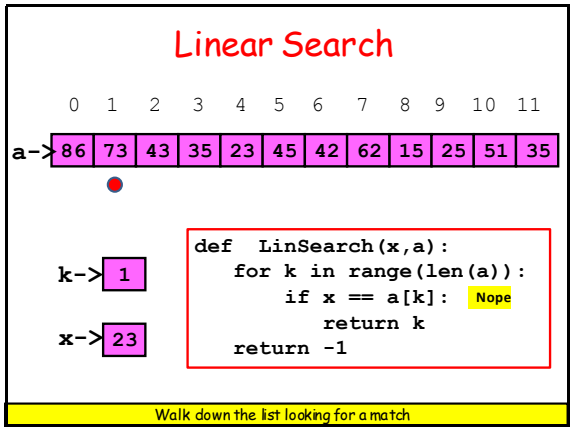
k-> 0

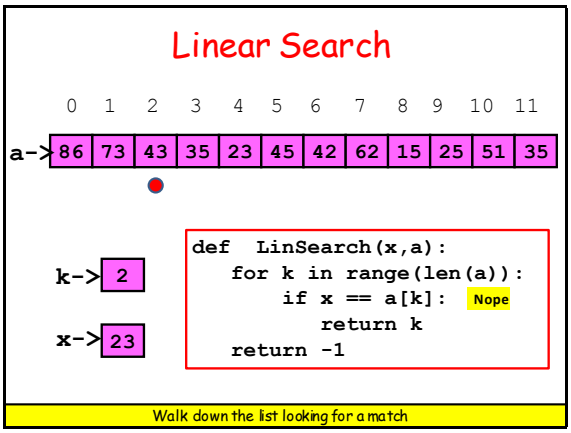
x-> 23

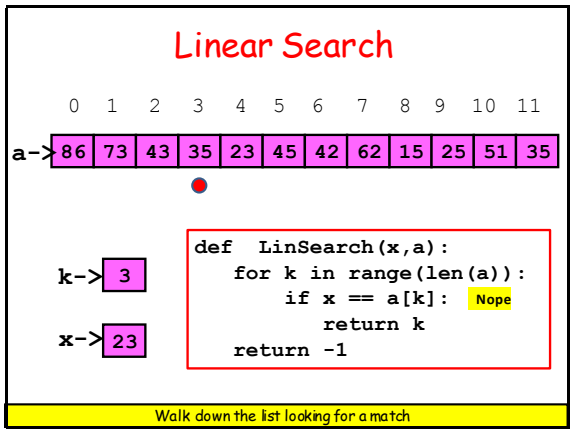
```
def  LinSearch(x,a):
    for k in range(len(a)):
        if x == a[k]:
            return k
    return -1
```

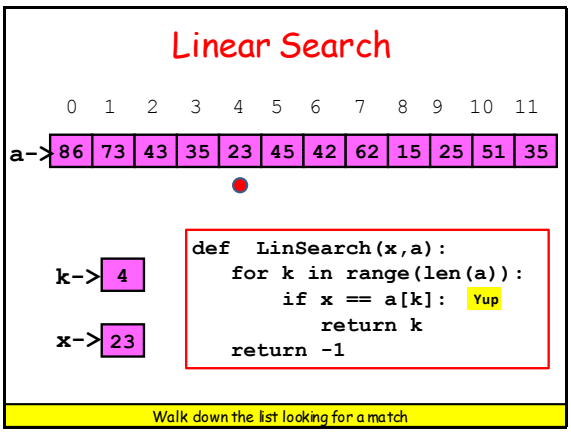
Walk down the list looking for a match

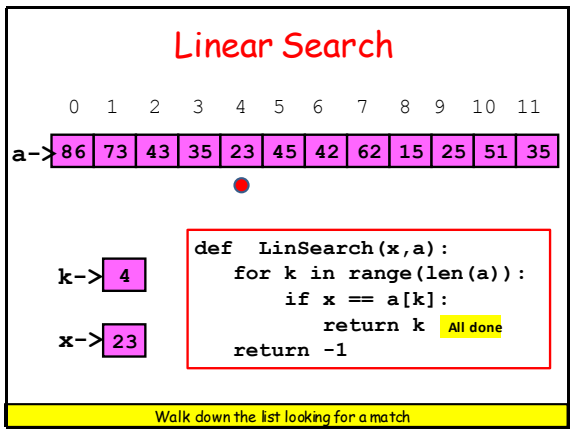

Linear Search
0 1 2 3 4 5 6 7 8 9 10 11
a-> 86 73 43 35 23 45 42 62 15 25 51 35
k-> 0
x-> 23
def LinSearch(x,a):
for k in range(len(a)):
if x == a[k]: Nope
return k
return -1
Walk down the list looking for a match


Linear Search
0 1 2 3 4 5 6 7 8 9 10 11
a-> 86 73 43 35 23 45 42 62 15 25 51 35
k-> 1
x-> 23
def LinSearch(x,a):
for k in range(len(a)):
if x == a[k]: Nope
return k
return -1
Walk down the list looking for a match


Linear Search
0 1 2 3 4 5 6 7 8 9 10 11
a-> 86 73 43 35 23 45 42 62 15 25 51 35
k-> 2
x-> 23
def LinSearch(x,a):
for k in range(len(a)):
if x == a[k]: Nope
return k
return -1
Walk down the list looking for a match


Linear Search
0 1 2 3 4 5 6 7 8 9 10 11
a-> 86 73 43 35 23 45 42 62 15 25 51 35
k-> 3
x-> 23
def LinSearch(x,a):
for k in range(len(a)):
if x == a[k]: Nope
return k
return -1
Walk down the list looking for a match


Linear Search
0 1 2 3 4 5 6 7 8 9 10 11
a-> 86 73 43 35 23 45 42 62 15 25 51 35
k-> 4
x-> 23
def LinSearch(x,a):
for k in range(len(a)):
if x == a[k]: Yup
return k
return -1
Walk down the list looking for a match


Linear Search
0 1 2 3 4 5 6 7 8 9 10 11
a-> 86 73 43 35 23 45 42 62 15 25 51 35
k-> 4
x-> 23
def LinSearch(x,a):
for k in range(len(a)):
if x == a[k]:
return k All done
return -1
Walk down the list looking for a match

Linear Search: No Match Case

	0	1	2	3	4	5	6	7	8	9	10	11
a->	86	73	43	35	23	45	42	62	15	25	51	35

k-> 11

x-> 7

```
def  LinSearch(x,a):
   for k in range(len(a)):
       if x == a[k]:   Nope
          return k
   return -1
```

Walk down the list looking for a match

Linear Search: No Match Case

	0	1	2	3	4	5	6	7	8	9	10	11
a->	86	73	43	35	23	45	42	62	15	25	51	35

x-> 7

```
def  LinSearch(x,a):
   for k in range(len(a)):
       if x == a[k]:
          return k
   return -1   Yup
```

Return -1 if no match

Linear Search: While Implementation

```
def  LinSearchW(x,a):
   k=0
   while k<len(a)  and a[k]!=x:
       k+=1
   if k==len(a):
       return -1
   else:
       return k
```

Binary Search

Now we assume that the list to be searched is sorted from little to big.

```
a = [10,20,40,60,90]

a = ['brown','dog','fox','lazy','quick','the']
```

Back to Using Phone Books

The Ithaca phone book has 10,000+ entries.

The Manhatten phone book has 1,000,000+ entries. But it does not take 100 x longer to look something up. Why?

Key Idea: Repeated Halving

To find Derek Jeter's number…

```
B = phone book
while (B is longer than 1 page):
   1. P = middle page of B
   2. Let Q be the first name on P
   3. if  'Jeter" comes before Q:
          Rip away the 2nd half of B
      else:
          Rip away the 1st half of B.

Scan remaining page P line-by-line for 'Jeter'
```

What Happens to Phone Book Length?

```
Original:      3000 pages
After 1 rip:   1500 pages
After 2 rips:   750 pages
After 3 rips:   375 pages
After 4 rips:   188 pages
After 5 rips:    94 pages

After 12 rips:    1 page
```

Binary Search

The idea of repeatedly halving the size of the "search space" is the main idea behind the method of binary search.

An item in a sorted array of length n can be located with approximately $\log_2 n$ comparisons.

$\log_2 8 = 3$ $\log_2 64 = 7$ $\log_2 2^{**}k = k$

What is $\log_2(n)$?

n	ceil($\log_2$(n))
10	4
100	7
1000	10
10000	14
100000	17
1000000	20

BinSearch: The Spec

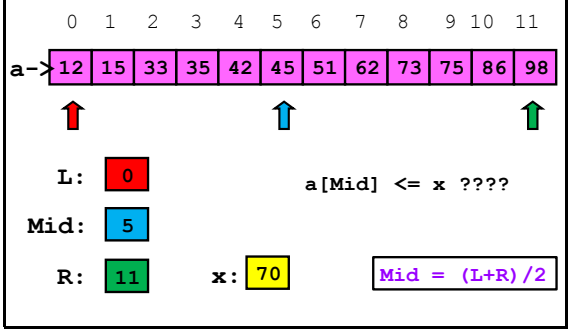
```
def BinSearch(x,a):
    """ Returns an int k with the
    property that a[k]==x is True.
    If no such k exists, then
    k==-1.

    PreC: a is a nonempty list of
    ints that is sorted from smallest
    to largest. x is an int.
    """
```

Example: Does this List have an Element With Value Equal to 70?

0	1	2	3	4	5	6	7	8	9	10	11
12	15	33	35	42	45	51	62	73	75	86	98

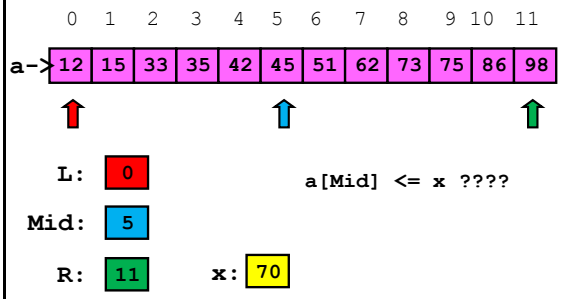
Let's Look For x in a



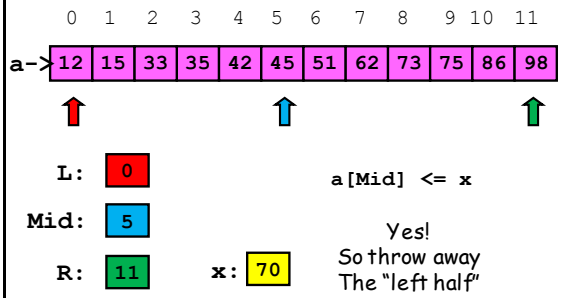
The Midpoint Computations

L	R	(L+R)/2
0	11	5
2	6	4
1	100	50

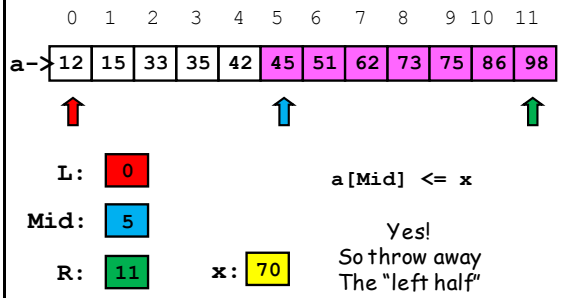
Let's Look For x in a



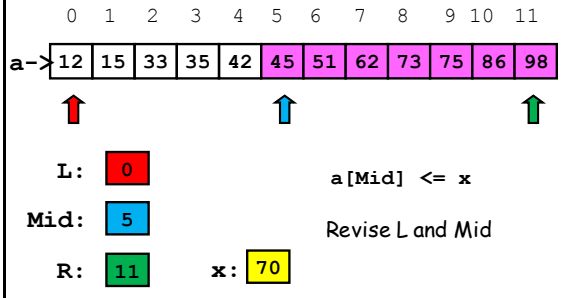
Let's Look For x in a



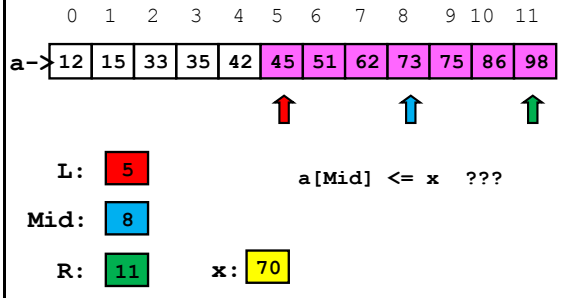
Let's Look For x in a

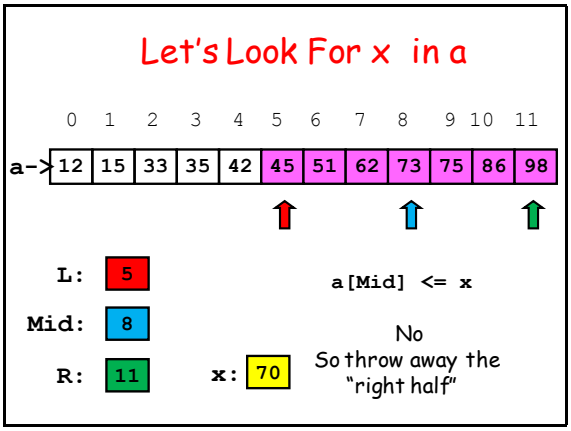


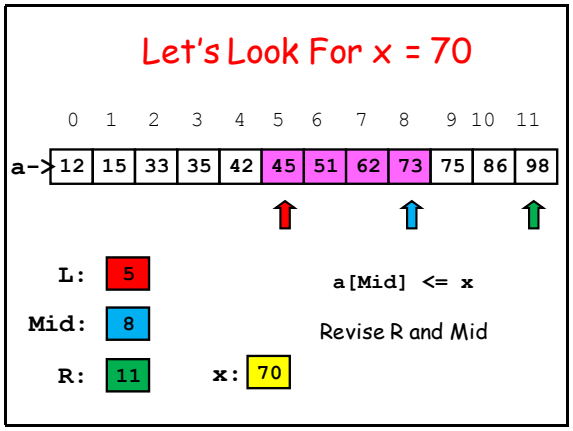
Let's Look For x in a

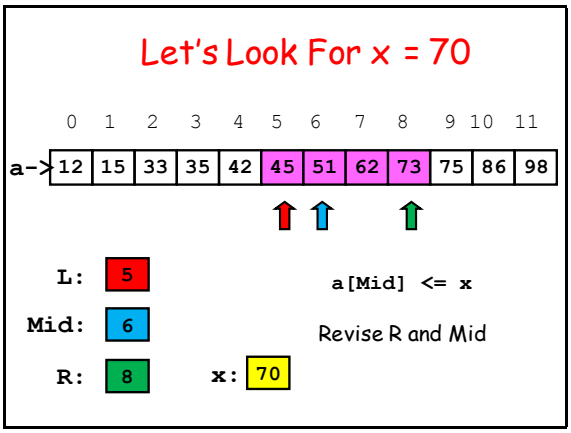


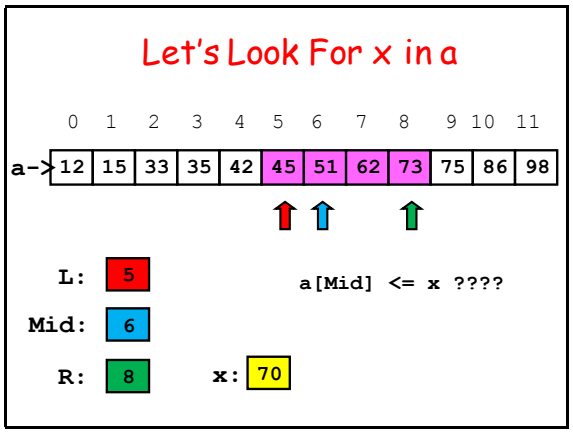
Let's Look For x in a

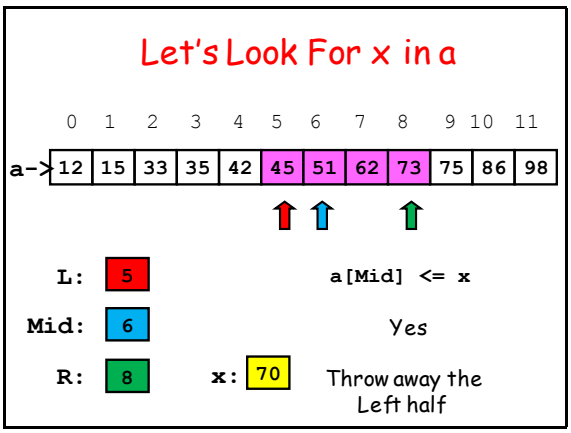


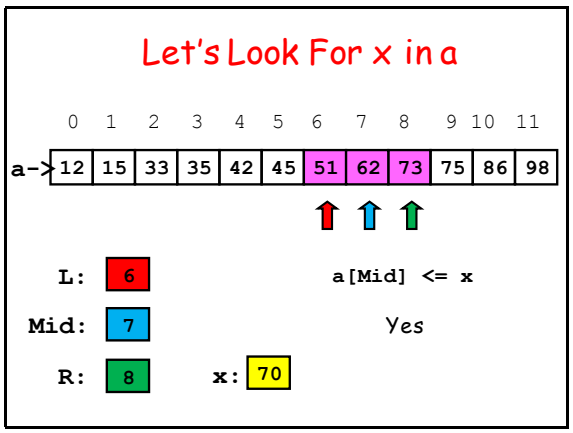

Let's Look For x in a
0 1 2 3 4 5 6 7 8 9 10 11
a-> 12 15 33 35 42 45 51 62 73 75 86 98
L: 5
Mid: 8
R: 11
x: 70
a[Mid] <= x
No
So throw away the "right half"


Let's Look For x = 70
0 1 2 3 4 5 6 7 8 9 10 11
a-> 12 15 33 35 42 45 51 62 73 75 86 98
L: 5
Mid: 8
R: 11
x: 70
a[Mid] <= x
Revise R and Mid

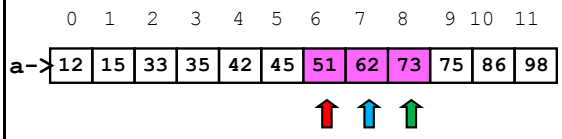

Let's Look For x = 70
0 1 2 3 4 5 6 7 8 9 10 11
a-> 12 15 33 35 42 45 51 62 73 75 86 98
L: 5
Mid: 6
R: 8
x: 70
a[Mid] <= x
Revise R and Mid


Let's Look For x in a
0 1 2 3 4 5 6 7 8 9 10 11
a-> 12 15 33 35 42 45 51 62 73 75 86 98
L: 5
Mid: 6
R: 8
x: 70
a[Mid] <= x ????


Let's Look For x in a
0 1 2 3 4 5 6 7 8 9 10 11
a-> 12 15 33 35 42 45 51 62 73 75 86 98
L: 5
Mid: 6
R: 8
x: 70
a[Mid] <= x
Yes
Throw away the Left half


Let's Look For x in a
0 1 2 3 4 5 6 7 8 9 10 11
a-> 12 15 33 35 42 45 51 62 73 75 86 98
L: 6
Mid: 7
R: 8
x: 70
a[Mid] <= x
Yes

Let's Look For x in a

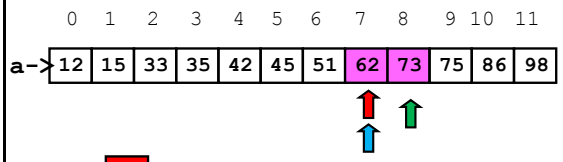


L: 6
Mid: 7
R: 8
x: 70

a[Mid] <= x

Throw away the left half

Let's Look For x in a



L: 6
Mid: 7
R: 8
x: 70

Done! At this point we just compare **x** with `a[L]` and `a[L+1]`.

What We Just Did

```
L = 0
R = len(a)-1
while R-L > 1:
    # a[L]<=x<=a[R]
     Mid = (L+R)/2
     if x <= a[mid]:
         R = Mid
     else:
         L = Mid
```

A Loop Invariant

Note that a[L]<=x<=a[R] remains True throughout the loop

What We Just Did

```
L = 0
R = len(a)-1
while R-L > 1:
    # a[L]<=x<=a[R]
     Mid = (L+R)/2
     if x <= a[mid]:
         R = Mid
     else:
         L = Mid
```

What is the situation when the loop terminates?

What We Just Did

```
L = 0
R = len(a)-1
while R-L > 1:
    # a[L]<=x<=a[R]
     Mid = (L+R)/2
     if x <= a[mid]:
         R = Mid
     else:
         L = Mid
```

R-L<=1 implies R = L+1

After the Loop Ends

This is True: `a[L]<=x<=a[L+1]`

After the Loop Ends


```
if x==a[L]:
   return L
elif x==a[L+1]:
   return L+1
else:
   return -1
```

Measuring Execution Time

We now have two ways to search a list:

```
LinSearch(x,a)
BinSearch(x,a)
```

Intuition: BinSearch much faster.

Can we quantify this with a "stop watch"?

The `timeit` Module

This module can be used to time how long it takes to execute a chunk of code.

Typical chunk = some function of interest.

This is called benchmarking.

Benchmarking

Let's benchmark `LinSearch(x,a)` and `BinSearch(x,a)`.

Compare how long it takes when `len(a)` equals 1000, 10000, 100000, and 1000000.

Our intuition tells us that as `len(a)` increases, `BinSearch` will be dramatically faster.

BinSearch vs LinSearch

n	tBin	tLin	tLinW
1000	0.0007	0.0064	0.0119
10000	0.0009	0.0668	0.1203
100000	0.0011	0.8296	1.2082
1000000	0.0015	17.7388	13.9341

```
tBin  = time for BinSearch
tLin  = time for LinSearch (for loop version)
tLinW = time for LinSearch (while-loop version)
```

BinSearch vs LinSearch

n	tLin/tBin
1000	9
10000	74
100000	754
1000000	7095

Reporting ratios is more illuminating since we do not really care about the time units in this informal comparison

Using the `timeit` Module

We show how this module was use to get the results on the previous slides.

Our LinSearch vs BinSearch example is very typical: is one function faster than another?

A Benchmarking Framework

```
from timeit import *
S = """
    Set-up code
"""
B = """
    Code to Benchmark
"""
p = 10; m = 100
t = min(Timer(B,setup=S).repeat(p,m))
```

Yes, these are doc strings.

The Set-Up and Bench Codes

```
from random import randint as randi
from ShowSearch import BinSearch
n = 10000
s = [randi(0,10*n) for i in range(n)]
s.sort()
x = s[n/2]
```

```
k=BinSearch(x,s)
```

The set-up code is run once.
It is not timed.
It just sets up the code to be timed.

A Benchmarking Framework

```
from timeit import *
S = """
    Set-up code
"""
B = """
    Code to Benchmark
"""
p = 10; m = 100
t = min(Timer(B,setup=S).repeat(p,m))
```

An "experiment" consists of running the blue code m times.

The stopwatch will time how long it takes to do one experiment

Larger values necessary if the blue code executes very quickly

A Benchmarking Framework

```
from timeit import *
S = """
    Set-up code
"""
B = """
    Code to Benchmark
"""
p = 10; m = 100
t = min(Timer(B,setup=S).repeat(p,m))
```

Timer returns a length-p list. Each element is the stopwatch time for 1 experiment

This helps control for other stuff that may be running on your computer.

A Benchmarking Framework

```
from timeit import *
S = """
    Set-up code
"""
B = """
    Code to Benchmark
"""
p = 10; m = 100
t = min(Timer(B,setup=S).repeat(p,m))
```

In general, it is best to take the mininum as the most reliable. The benchmark time is assigned to t

This helps control for other stuff that may be running on your computer.

Why Benchmarking is Important

Confirms/refutes what our intuition might say about efficiency.

Makes us sensitive to the various issues that affect efficiency.

Steers us away from simplistic comparisons of different methods that can be used on the same problem.